`GrowFundCurv1D`: computation with MATLAB of one-dimensional manifolds in maps

Dana C'Julio, Bernd Krauskopf and Hinke M. Osinga

1 Introduction

The package `GrowFundCurv1D` is an implementation in Matlab of the algorithm presented in [1] for the numerical computation of one-dimensional (un)stable manifolds and intersection points in maps, based on the manifold growth algorithm from [3]. This manual serves as a practical guide for utilising the package `GrowFundCurv1D` without delving into extensive algorithmic details. We refer to [1] for a comprehensive discussion of the algorithm and its accuracy constraints, as well as more details on its performance. In case you use `GrowFundCurv1D` in your own research, please, give credit by citing [1].

The package `GrowFundCurv1D` has been designed for three-dimensional maps, and uses an initial short segment of a stable or unstable manifold associated with a fixed or periodic point. This initial segment can be obtained either from the linear approximation, given by the eigenvector associated with the respective stable or unstable eigenvalue, or from a previous continuation. The algorithm first identifies the last fundamental domain on the initial segment, based on which the iterative process of growing the manifold starts. As a post-processing step, the algorithm computes the intersection points of the computed manifold with a pre-specified plane as an ordered set.

The package `GrowFundCurv1D` comprises a series of routines, which are available in the folder `GrowFundCurv1D_code/` that can be downloaded from `https://www.math.auckland.ac.nz/~hinke/preprints/cko_algorithm.html`. The folder contains the Matlab script file `GrowFundCurv1D_demo.m` that demonstrates the algorithm with a specific example, the comprehensive step-by-step manual `GrowFundCurv1D_manual.pdf` (this file), the data used to obtain the first fundamental domain of the manifold, and the folder `GrowFundCurv1D_functions/` with required functions.

The demo `GrowFundCurv1D_demo.m`, presented in the next section, was tested using Matlab [version 9.12 (R2022a)]. This example computes a one-dimensional stable manifold of a fixed point for a three-dimensional Hénon-like map, as defined in [1]. Note that, with appropriate changes to the accuracy settings, the algorithm can accurately compute manifolds not only for the fixed points of the map itself, but also for up to its fourth iterate, without losing resolution.

2 Demo

We include the demo script `GrowFundCurv1D_demo.m` with the package `GrowFundCurv1D`, which computes and plots the stable manifold of a fixed point with the default accuracy parameters defined in Section 6. Specifically, the map is the three-dimensional Hénon-like map from [1], the parameters are $\alpha = 4.2$, $\beta = 0.3$ and $\xi = 1.2$, and the computed manifold is $W^s(p^-)$ associated with the fixed point p^- that has negative x- and y-coordinates; see Figure 1.

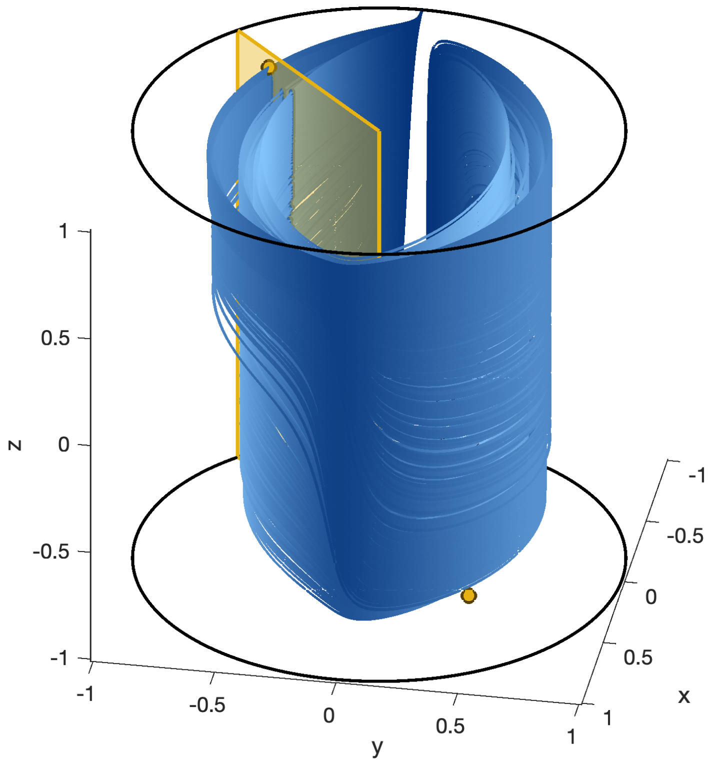

Figure 1: Output of the demo script `GrowFundCurv1D_demo.m` that computes the one-dimensional stable manifold of a fixed point for the three-dimensional Hénon-like map from [1]. Shown are, in compactified (x, y, z)-space, the fixed points (golden dots), the stable manifold (blue curve) of the top-left fixed point, the half-plane Σ (yellow surface) and the intersection points with Σ (blue dots). Also shown are the boundary circles of the compactification.

To run the demo, ensure all files and folders in `GrowFundCurv1D/` are in the search path of the active MATLAB directory. The demo is then run by typing `GrowFundCurv1D_demo` into the command window of MATLAB. The following tasks are then performed:

- the paths to the necessary functions are added;
- the initial segment of the manifold is loaded and saved;
- the map information is specified and stored in the structure `opts`;
- a comprehensive data structure is generated by the function `init_manif` to be used and modified by the core function `GrowFundCurv1D`;
- `GrowFundCurv1D` is called to compute the manifold up to the specified number of steps;
- the function `intersect_plane` is called to find the intersection points with the pre-specified plane Σ;

- the function manifplot is called to plot the manifold, the plane and its intersection points, as shown in figure 1.

3 Formulation of the method

We consider a diffeomorphism
$$f : \mathbb{R}^n \to \mathbb{R}^n$$
that has a hyperbolic fixed point $p = f(p) \in \mathbb{R}^n$ with a single real and positive unstable eigenvalue $\lambda^u > 1$. According to the Stable Manifold Theorem [4], the unstable manifold

$$W^u(p) = \left\{ x \in \mathbb{R}^2 \;\middle|\; f^{-k}(x) \to p \text{ as } k \to \infty \right\}$$

is then a one-dimensional smooth curve that is tangent at p to the unstable eigenvector $\mathbf{v}^u$. Similarly, the stable manifold $W^s(p)$ of p is an $(n-1)$-dimensional manifold, but we do not consider it here.

As is common in the field, we formulate our algorithm for the computation of $W^u(p)$ for notational convenience. Since f is a diffeomorphism, any one-dimensional stable manifold can simply be computed as the unstable manifold of the inverse f^{-1} of f. We also assume that λ^u is a positive eigenvalue, which means that f is orientation preserving on $W^u(p)$ and, hence, consists of two branches (on either side of p) that are each invariant under f.

4 Structure of the algorithm

The algorithm is organised into four steps, each serving a distinct purpose.

Step 1: In this initial step, the user defines the system and its fixed/periodic points. This is accomplished by creating a class of functions, referred to as `StdHenon3D` in this guide. A more detailed explanation of the structure of this class is provided in Section 5.

Step 2: In the second step, the user provides parameter values, accuracy parameters, the required number of iterations, and sets up the initial segment of the manifold. This information is organised within a structure named `opts`, the formulation of which is explained in Section 6. The structure `opts` is passed as an argument to the function init_manif, which generates a comprehensive data structure called `manif`. Then `manif` is subsequently used in Step 3. Details of the `manif` data structure are outlined in Section 7.

Step 3: The third step is the core implementation of the algorithm, as described in [1]. It involves the function GrowFundCurv1D, which iterates an initial segment of the manifold of the fixed or periodic point, ultimately producing the complete manifold up to a pre-specified iteration number. The structure `manif` is updated with the information from the output given by GrowFundCurv1D.

Step 4: In the final step, the algorithm computes the ordered list of intersection points on the manifold with a pre-specified plane, and then stores these in the structure `manif`. This is done with the function intersect_plane, as discussed in Section 8. It is important to note that, for our specific example, we use a half-plane that contains the origin and is perpendicular to the (x, y)-plane.

To get a better grasp of the algorithm, Section 2 provides a walk-through of the computations for the example of the three-dimensional Hénon-like map.

5 Step 1: Defining the system

All the functions that need to be provided by the user to define the system are consolidated into a single class, which we refer to as `StdHenon3D` in this guide; this class name can be chosen arbitrarily and `StdHenon3D` is simply used as an example.

Typically, the user is required to provide a few essential functions within this class, including the definition of the diffeomorphism f, its inverse f^{-1}, and the fixed point associated with the manifold to be computed. In addition to the primary fixed point, the user has the flexibility to include any number of fixed points of the map for future reference.

A detailed explanation of these functions is provided below. For a guide to the format and semantics of the following functions within this class, please, also refer to Table 1.

```
function map_points = ff(points, opts)
function map_points = ff_inv(points, opts)
function fixpinfo = fixpoints(opts)
function comp_points = compactify(points)
function decomp_points = decompactify(points)
function outpoints = mapping(inpoints, stab, opts)
```

Note that all functions with the structure `opts` as input entry use the same options; full details of this structure are explained in Section 6.

Functions ff and ff_inv

The functions ff and ff_inv define the diffeomorphism f and the inverse f^{-1} used in the computation. They are called by the function mapping to compute the images (preimages) of points on the unstable (stable) manifold. These functions have two arguments:

functions		
name	**user-provided**	**content**
ff	**yes**	diffeomorphism f
ff_inv	**yes**	diffeomorphism f^{-1}
fixpoints	**yes**	fixed points
compactify	**yes**	compactification of the space
decompactify	**yes**	inverse of the compactification
mapping	**no**	applies f or f^{-1}

Table 1: Summary of the class of functions passed to the algorithm.

- `points`: a structure containing the fields `x`, `y` and `z` that are the x-, y- and z-coordinates of the points on which to apply the function f or f^{-1}.
- `opts`: options; only the field `par` from `opts` is passed to ff or ff_inv, which contains the parameter values needed for the computation.

Each function returns a structure named `map_points`, which contains the fields `x`, `y` and `z` representing the x-, y- and z-coordinates of the (pre)images of the input structure `points`.

Function fixpoints

This function returns the fixed points of the system and contains their explicit formulas. It is mandatory to include at least the fixed point associated with the manifold to be computed, but additional fixed points of the system can be added for reference. This function has one argument:

- `opts`: options; the field `par` from `opts` is used for computing the fixed points.

This function returns a structure named `fixpinfo` with the explicit coordinates of the fixed point for the chosen parameter values.

Functions compactify and decompactify

For our example map `StdHenon3D`, we use compactified coordinates. Hence, we introduce two functions compactify and decompactify that define the coordinate transformations of the compactification and its inverse, respectively. They are integrated into the function mapping and the user needs to change them to the correct transformation, or set them as the identity if they work in non-compactified coordinates. Each function has one argument:

- `points`: a structure containing the fields `x`, `y` and `z`, which are the x-, y- and z-coordinates of the points to compactify or decompactify.

The outputs of functions compactify and decompactify are structures with names `comp_points` and `decomp_points`, respectively.

Function mapping

The function mapping is used by the main routine `GrowFundCurv1D` to compute the (pre)images of a set of points. Typically, the user does not need to modify this function. It contains three arguments:

- `inpoints`: a structure containing the fields `x`, `y` and `z`, which are the x-, y- and z-coordinates of the points to which the map is applied.
- `stab`: a string indicating the type of manifold being computed. It is set to `'Smanifold'` for a stable manifold and `'Umanifold'` for an unstable manifold. This information is not set by the user; it is an output of the function init_manif from Section 7.
- `opts`: options; there are two fields from `opts` passed to mapping, namely, `mapiter`, which contains the number of iterations of the map, and `par` comprising the parameter values.

The function mapping applies the following three subfunctions:

- the function decompactify applies the inverse of the compactification to the structure of points to obtain the original coordinates;
- either ff or ff_inv based on the type of manifold;
- the function compactify to return to compactified coordinates.

The function is iterated a pre-specified number of times, given by the field `mapiter` in `opts`. It returns a structure called `outpoints`, containing the fields `x`, `y` and `z` representing the x-, y- and z-coordinates of the (pre)image of the input structure `inpoints`.

6 Step 2: Set-up of parameters and initial segment

All the information needed for the computation of the manifold is stored by the user in a structure named `opts`. Table 2 shows a summary of all the fields of `opts`.

`opts.thesystem=StdHenon3D;`
specifies the user-defined class that defines the diffeomorphism and fixed points. This structure serves the purpose of selecting the appropriate set of functions when the manifold computation is in progress.

`opts.par=struct('a', 4.2, 'b', 0.3, 'xi', 1.2);`
sets the names and values of the parameters of the system. The names of the param-

opts			
field	subfields	type	definition
`thesystem`		class	Name of the class where the system is defined
`par`	`{ par1, par2, ...}`	$\mathbb{R}$	Values of the parameters
`name_fixpoint`		string	Name of the fixed point associated to the manifold
`branch`		string	Name of the branch of the manifold
`init_segment`	`{ x, y, z }`	vector	The (x, y, z)-coordinates of the initial segment
`funditer`		$\mathbb{N}$	Number of iterations of the fundamental domain
`mapiter`		$\mathbb{N}$	Number of iterations of the map
`angle`		$[-\pi, \pi]$	Angle of the plane when computing intersection points
`accpar` (optional)	`{alphamax, deltalphamax, deltamin, deltamax}`	$\mathbb{R}$	Accuracy parameters

Table 2: Options defined in the structure `opts` that initialise the setting needed to compute the (un)stable manifold.

eters have to be consistent with the ones defined in the system class `StdHenon3D` from Section 5.

```
opts.name_fixpoint='pmin';
```
identifies the name of the fixed point for which the manifold will be computed. This information is is passed to the output data and is solely for the user's reference. If the user does not need this information, they can leave it as an empty string.

```
opts.branch='pos';
```
identifies the particular branch of the computed manifold—here, `'pos'` refers to the branch that lies in the direction to the right of the x-coordinate of the fixed point; it is again solely for the user's reference and can be left as an empty string.

```
opts.init_segment=struct('x', data_x, 'y', data_y, 'z', data_z);
```
contains a structure that holds the coordinates of the starting segment of the manifold, from which the fundamental domain will be extracted. Typically, these coordinates are set as `'x'`, `'y'` and `'z'`. It is not recommended to change the names of these coordinates to ensure consistency with other functions. The entries `data_x`, `data_y` and `data_z` are vectors containing the (x, y, z)-coordinates of the initial segment of the manifold, respectively.

`opts.funditer=6;`
means that the fundamental domain is iterated a total of six times.

`opts.mapiter=2;`
allows the user to specify the iterate of the map that is applied to the fundamental domain. It is important to note that `opts.funditer` should be a multiple of `opts.mapiter`. In our example, the iteration process involves using the map $\mathcal{H}^2$ three times ($3 \times$ `opts.mapiter` $=$ `opts.funditer`).

`opts.angle=-3*pi/4;`
this field is only necessary when using the function `intersect_plane`, which computes the ordered intersection points of the manifold with a plane. In our example, we use a half-plane trough the origin that is perpendicular to the (x, y)-plane. It forms an angle $\theta \in [-\pi, \pi]$ with respect to the plane $\{y = 0\}$.

If θ is positive, the half-plane extends counter-clockwise from the positive x-axis. Conversely, if θ is negative, it extends clockwise from the positive x-axis. In our specific example, `opts.angle` specifies the half-plane $\Sigma = \{(x, y, z) \in \mathbb{R} \mid x = y \text{ and } x < 0\}$.

`opts.accpar`
contains the parameters that control the accuracy of the computations. These parameters are:

- The maximum $\alpha_{\max}$ of the angle allowed between triplets of mesh points;
- The curvature parameter $(\Delta\alpha)_{\max}$;
- The maximum $\Delta_{\max}$ and minimum $\Delta_{\min}$ of the permitted distances between mesh points.

Setting these parameters is optional; their default values are:

```
alphamax=0.3;
deltalphamax=0.001;
deltamin=0.000001;
deltamax=0.01;
```

For example, if the user only wants to change $\Delta_{\max}$, this must be done before calling the function `init_manif`, as follows:

```
opts.accpar.deltamax=0.05;
```

Unless accuracy parameter are defined by the user, the function `init_manif` (see Section 7) will use the default settings, which are selected to ensure that the algorithm can accurately compute manifolds not only for the fixed points of the map $\mathcal{H}$ itself but also for up to its fourth iterate without losing resolution. We refer to [1] for a comprehensive understanding of these accuracy constraints.

manif			
field	subfields	type	definition
`name`		string	Name of the manifold
`orientability`		string	Orientation properties of the manifold
`fixp`	`{ pmin, eigsys }`	string	The fixed point of the manifold and its eigensystem
`stab`		string	Stability of the manifold (the options are `'Umanifold'` or `'Smanifold'`)
`points`	`{ x, y, z, arc }`	vector, $\mathbb{R}$	Coordinates and arclength of the manifold
`inf_sys`	`{ par, fixp }`	$\mathbb{R}$, string	Parameter values and the fixed points
`growinf`	`{ mapiter, funditer, alphamax, deltalphamax, deltamin, deltamax }`	$\mathbb{R}$ or $\mathbb{N}$	Information extracted from the structure `opts` from Section 6
`runinf`	see Table 4		Additional subfield added by the core function `GrowFundCurv1D`

Table 3: Names and definitions of the fields and subfield in the structure `manif`.

7 Step 3: the core of the package `GrowFundCurv1D`

The core of the package `GrowFundCurv1D` starts from a structure that is constructed from the user-provided options, as described in Section 6. However, rather than only passing the structure `opts` as an argument to the main routine `GrowFundCurv1D` of the algorithm, the function `init_manif` is called instead, which is not typically modified by the user:

```
manif = init_manif(opts);
```

The output of `init_manif` is a structure called `manif` that stores the computed manifold and all necessary information, which is summarised in Table 3. In particular, `manif` contains the subfield `growinf` with the accuracy parameters and number of iterations employed in the computation.

Rather than modify the input `manif`, the function `GrowFundCurv1D` takes `manif` as input and returns an updated copy of this structure with additional fields to it:

```
manif=GrowFundCurv1D(manif, opts);
```

More precisely, `GrowFundCurv1D` updates the field `manif.points` with the computed manifold. It also adds an extra field called `runinf` to `manif.growinf`, which contains diagnostics related to the computation of the manifold; see Table 4.

manif.growinf.runinf		
subfield	type	definition
rem_deltamin	$\mathbb{N}$	Number of points removed due to being with distance $\Delta_{\min}$
rem_nan	$\mathbb{N}$	Number of points removed due to presence of NaN
rem_inf	$\mathbb{N}$	Number of points removed from infinity (compactification)
add_alphamax	$\mathbb{N}$	Number of points added because $\alpha_{\max}$ is too large
add_deltamax	$\mathbb{N}$	Number of points added because $\Delta_{\max}$ is too large
add_deltalphamax	$\mathbb{N}$	Number of points added because $(\Delta\alpha)_{\max}$ is too large
npoints_initial_final	$\mathbb{N}$	Number of initial and final points
arc_initial_final	$\mathbb{R}$	Initial arclength and final arclength
time	$\mathbb{R}$	Time spent on the computation

Table 4: Names and definitions of the subfield manif.growinf.runinf

manif.inter		
field	type	definition
angle	$[-\pi, \pi]$	the angle of the plane, defined in Section 6
plane	string	Name of the half-plane Σ
idx	$\mathbb{N}$	Vector with the indices of the intersection points

Table 5: Names and definitions of the subfield manif.inter.

8 Step 4: post-processing the results

Once the manifold has been computed, the final step is to process the data for analysis and visualisation purposes. We implemented the function intersect_plane to determine the (ordered) intersection points of the manifold with a pre-specified plane Σ, as defined in opts.angle; see Section 6. The function intersect_plane again does not modify the input manif; rather it takes manif as input and returns an updated copy of this structure with additional fields to it:

```
manif=intersect_plane(manif, opts);
```

The output is stored in the field manif.inter that is added to the structure manif; its subfields are shown in Table 5. The intersection points are computed by detecting sign changes with respect to a normal vector of the plane Σ and then finding the corresponding intersection points with cubic spline interpolation from nearby points. Each intersection point is then inserted at the correct position into the field manif.points, with a flag identifying it as such. Specifically, the flag is the index of such an inserted point and is stored in the subfield inter.idx. The ordering of the intersection points is induced by the the fact that the manifold is parameterised by arclength.

The function manifplot plots the results obtained from GrowFundCurv1D and intersect_plane. The simple command

```
h=manifplot(manif)
```

extracts the relevant information stored in the structure manif and plots a three-dimensional image. Here, we make use of the function color_line3 by [5], which transforms a one-dimensional object into a surface that can be assigned a colour scheme controlled by a colormap. For our specific implementation, we define a colormap based on the distances to the vertical line $(x, y) = (0, 0)$, generating an illusion of light that enhances the visual representation of the results.

References

[1] D. C'Julio, B. Krauskopf, and H.M. Osinga. Computing parametrised large intersection sets of 1D invariant manifolds: a tool for blender detection. Preprint available from `https://www.math.auckland.ac.nz/~hinke/preprints/cko_algorithm.html`, 2023.

[2] D. Hobson. An efficient method for computing invariant manifolds of planar maps. *Journal of Computational Physics*, 104(1): 14–22, 1993.

[3] B. Krauskopf and H. M. Osinga. Growing 1D and quasi-2D unstable manifolds of maps. *Journal of Computational Physics*, 146(1): 404–419, 1998.

[4] J. Palis and W. de Melo. *Geometric Theory of Dynamical Systems*. Springer-Verlag, New York, 1982.

[5] G. Stillfried. 3D colored line plot (`https://www.mathworks.com/matlabcentral/fileexchange/23566-3d-colored-line-plot`). MATLAB Central File Exchange. Retrieved 11 September, 2023.